НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В. Коваль
(підпис) (ініціали, прізвище)
“___”_____2019р.

# ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 6.050101 “ Комп’ютерні науки ”

на тему: «Розробка програмного агента моніторингу та управління системи вентиля»

Виконав: студент IV курсу, групи ТМ-51

Руденко Микола Володимирович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник старший викладач Мірошниченко І.В. _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Рецензент_____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

**Національний технічний університет України**
**“Київський політехнічний інститут імені Ігоря Сікорського”**

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 6.050101 “ Комп’ютерні науки”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)

”___”____2019р.

# ЗАВДАННЯ
**на дипломну роботу студенту**
Руденко Миколі Володимировичу
(прізвище, ім’я, по батькові)

1. Тема роботи: «Розробка програмного агента моніторингу та управління системи вентиля»
керівник роботи Мірошниченко Іван Володимирович
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”__”___ 201__р. № __

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи мова програмування Swift, середовище Xcode, сервіс Firebase

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати норми складу повітря у приміщенні, створити алгоритм енергоефективного використання вентиляційної установки, розробити мобільний додаток для моніторингу та інтелектуального керування вентиляційною установкою

5. Перелік ілюстративного матеріалу архітектура системи, графічне представлення інтерфейсу, приклади роботи програмного модулю
6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| | | | |

7. Дата видачі завдання ”___”_____2018 р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів виконання дипломної роботи | Термін виконання етапів роботи | Примітки |
|---|---|---|---|
| 1. | Затвердження теми роботи | 09.10.17 | |
| 2. | Вивчення та аналіз задачі | 05.02 – 11.02.18 | |
| 3. | Розробка архітектури та загальної структури системи | 12.02 – 18.02.18 | |
| 4. | Розробка структур окремих підсистем | 18.02 – 26.02.19 | |
| 5. | Програмна реалізація системи | 26.02 – 6.03.19 | |
| 6. | Оформлення пояснювальної записки | 13.03 – 30.05.19 | |
| 7. | Захист програмного продукту | 18.05.19 | |
| 8. | Передзахист | 1.06.19 | |
| 9. | Захист | | |

Студент _____ Руденко М. В.
(підпис) (прізвище та ініціали,)

Керівник роботи _____ Мірошниченко І.В.
(підпис) (прізвище та ініціали,)

# АНОТАЦІЯ

Дипломну роботу виконано на 40 аркуш, вона містить 3 додатки та перелік посилань на використані джерела з 9 найменувань. У роботі наведено 20 рисунків.

Метою даної дипломної роботи є створення мобільного додатку для моніторингу та автоматичного керування вентиляційною установкою. Додаток має бути зручним в користуванні, мати зрозумілий користувачу інтерфейс та енергоефективно керувати установкою.

Ключові слова: енергоефективність, мобільний додаток, автоматичне керування, вентиляційна установка.

# ABSTRACT

The thesis is presented in 40 pages. It contains 3 appendixes and bibliography of 9 references. Twenty figures are given in the thesis.

The purpose of this thesis is to create a mobile application for monitoring and automatic control of the ventilation installation. The application should be user-friendly, user-friendly interface and energy efficient installation management.

Keywords: energy efficiency, mobile application, automatic control, ventilation installation.

# ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМОВЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

| | |
|---|---|
| ПЗ | програмне забезпечення |
| UI (User Interface) | інтерфейс користувача |
| Фреймворк | (англ. Framework, каркас, платформа, структура, інфраструктура) – інфраструктура програмних рішень, що полегшує розробку складних систем |
| IDE (Integrated Development Environment) | комп'ютерна програма, що допомагає програмістові розробляти нове програмне забезпечення чи модифікувати (удосконалювати) вже існуюче |
| GCC (GNU Compiler Collection) | набір компіляторів для різних мов програмування |
| LLVM (Low Level Virtual Machine) | універсальна система аналізу, трансформації і оптимізації програм, щореалізує віртуальну машину з RISC-подібними інструкціями |
| Interface Builder | середовище для розробки інтерфейса корситувача |
| Developer Tools | інструменти для розробки |
| Юніт-тестування | модульне тестування |
| RPC (Remote Procedure Call) | протокол, що дозволяє програмі, запущеній на одному комп'ютері, бути викликаною на іншому комп'ютері без написання безпосередньо коду для цієї операції |

# ВСТУП

В наш час все частіше постає питання економії електричної енергії при використанні електричних приладів. Також зі збільшенням обсягів виробництв в навколишньому середовищі, люди все більш звертають увагу на умови в приміщеннях де вони перебувають. Для цього в приміщеннях встановлюють вентиляційні системи з автоматичним керуванням.

Така система повинна надавати користувачу вибір режиму роботи, забезпечувати своєчасне постачання свіжого повітря та контролювати якість повітря.

Система аналізує показники отримані з датчиків встановлених у приміщенні, перевіряє отримані данні з нормами та розподіляє потужність по кімнатам в яких потребується вентиляція.

На даний момент існують системи керування вентиляційною системою які вмикаються за розкладом або примусово. Такі системи не є енергоефективними та не завжди можуть забезпечить необхідну якість повітря. Данні системи не слідкують за показниками, тому при ввімкненні не береться до уваги необхідність їх використання. Встановленого часу роботи за розкладом може бути не достатньо, або навпаки систему можуть працювати тоді коли це не є необхідним.

# 1 ПОСТАНОВКА ЗАДАЧІ

Метою даної дипломної роботи є реалізація мобільного додатку, що надає можливість моніторингу та автоматичного керування вентиляційною установкою.

Для реалізації додатку були проаналізовані наступні данні:

а) норми стану повітря у приміщенні;

б) способи перевірки стану вентиляційного фільтру;

в) способи комунікації додатку з сервісом Firebase;

Призначення програмного продукту полягає в:

а) автоматизації процесу розрахунку вентиляційної установки;

б) оптимізації процесу розподілення потужності системи;

в) уникненні зайвих витрат електроенергії;

г) забезпеченні швидкої взаємодії користувача з системою, з будь якої точки країни використовуючи мобільний додаток.

# 2 ЗАСОБИ РОЗРОБКИ

Одним із найважливіших завдань при розробці програмних продуктів є вибір таких засобів, які б полегшили роботу програміста, надавши всі необхідні інструменти для реалізації поставленого завдання, і дали б змогу отримати результат, який повністю задовольняє користувача.

## 2.1 Огляд середовища розробки

Середовище розробки Xcode є додатком для розробки програмних застосунків від компанії Apple. Додаток постійно оновлюється та супроводжується розробником. Безкоштовно середовище можна скачати у магазині додатків “Mac App Store” (рис. 2.1).

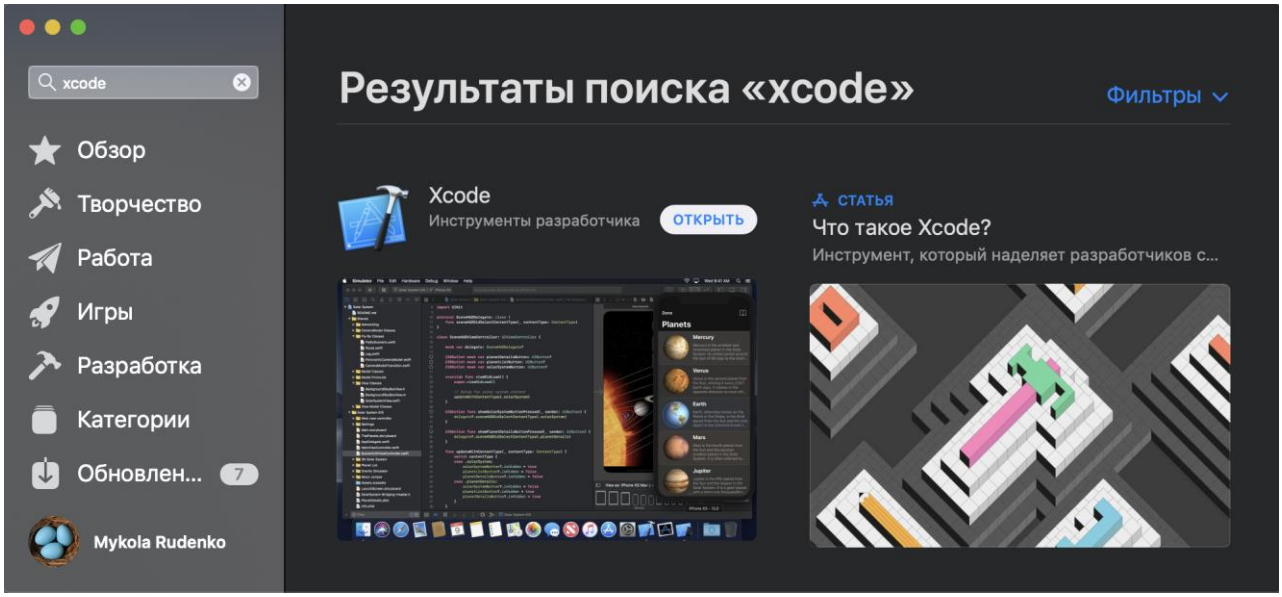


Рисунок 2.1 – Магазин додатків “Mac App Store”

Основними мовами програмування у середовищі є Swift, Objective-C та C++. Більш за все середовище використовується для створення мобільних додатків з

операційною системою iOS та десктопних додатків з операційною системою Mac OS. Для зручного використання Xcode має велику кількість можливостей:

- система для керування проектом та доданими модулями;
- середовище для написання коду продукту з автоматичними підказками та редагуванням коду;
- швидкий доступ до документації коду;
- емулятор пристроїв для тестування продуктів;
- засоби для швидкого редагування коду на рівні усього проекту;
- система для роботи з локальними та хмарними системами керування версіями;
- функція попереднього перегляду вигляду створеного програмного продукту враховуючи усі встановленні залежності та обмеження;
- можливості для налагодження роботи написаного коду програми;
- сучасна система будування проекта з тестуванням та відлагодженням перед запуском або випуском у реліз;
- засіб для відстеження навантажень на систему при тестуванні;
- можливість симуляції місця положення пристроїв для тестування продуктів з використанням GPS;
- інспектор підтримки коду на визначених операційних системах;
- вбудована система для контролю та керуванням вихідними даними;
- можливість об’єднання модулів на різних мовах програмування в одному проекті;
- середовище для налагодження відображення 3D елементів.

Середовище Storyboard спрощую створення інтерфейсу створюємого продукту. З його допомогою до інтерфейсу додаються кнопки, переходи між

екранами, підписи, елементи для відображення рисунків. Існує можливість створення нестандартних елементів інтерфейсу.

Нестандартні елементи інтерфейсу створені в Storyboard можна використовувати необмежену кількість раз у додатку. Елементи інтерфейсу можна змінювати, налаштовувати та закріпляти для автоматичного масштабування при необхідності. Кожен елемент інтерфейсу є екземпляром класу, та може бути переголошен як екземпляр іншого класу з наслідсвом параметрів та методів попереднього. Таким чином розробник індивідуалізує створюємий додаток. З елементами інтерфейсу можна працювати як і з середовища Storyboard, так і з коду підключивши їх (рис. 2.2).

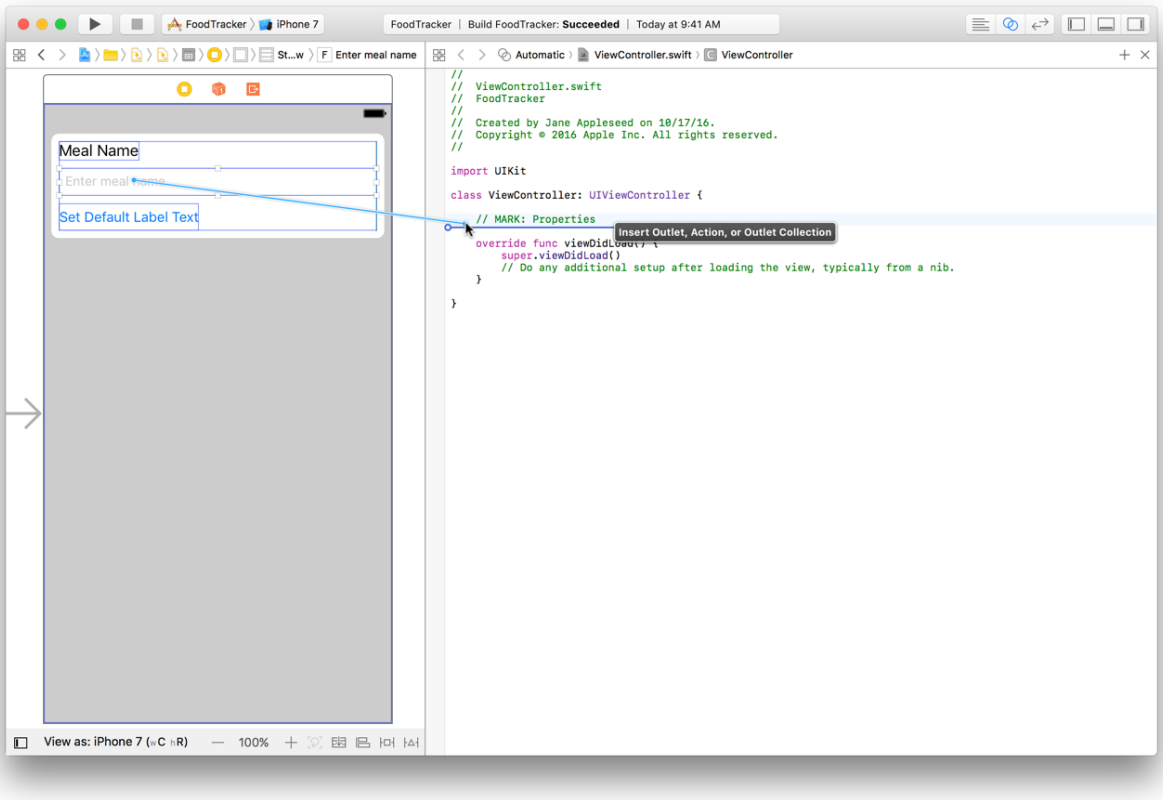


Рисунок 2.2 – З‘єднання елементів інтерфейсу з кодом

Використання коду для налаштування та керування елементами інтерфейсу надає більше можливостей. Використовуючи код при роботі з елементами зі

Storyboard можна перехоплювати дії елементу, та опираючись на цикл роботи керувати та контролювати процес роботи додатку. Для роботи з елементами інтерфейсу в коді необхідно використовувати бібліотеку “UIKit”.

Сучасні мови програмування, такі як Swift, спрощують процес написання коду. Більшість бібліотек мають методи необхідні для більшості проектів. Кожен розробник може використовувати їх, тим самим зменшити рас розробки, вартість проекту, та збільшити безпечність свого коду.  Безпечність коду є дуже важливою частиною програмування. Метою такого написання коду є забезпечення продуктивності роботи програмного продукту, запобігання викрадення або втрати даних, та уникнення збоїв програми. Для перевірки продуктивності роботи додатку розробник може користувати потужний інструмент для моніторингу продуктивності вбудований в Xcode поки застосунок є запущений на реальному пристрої або емулятори (рис. 2.3).

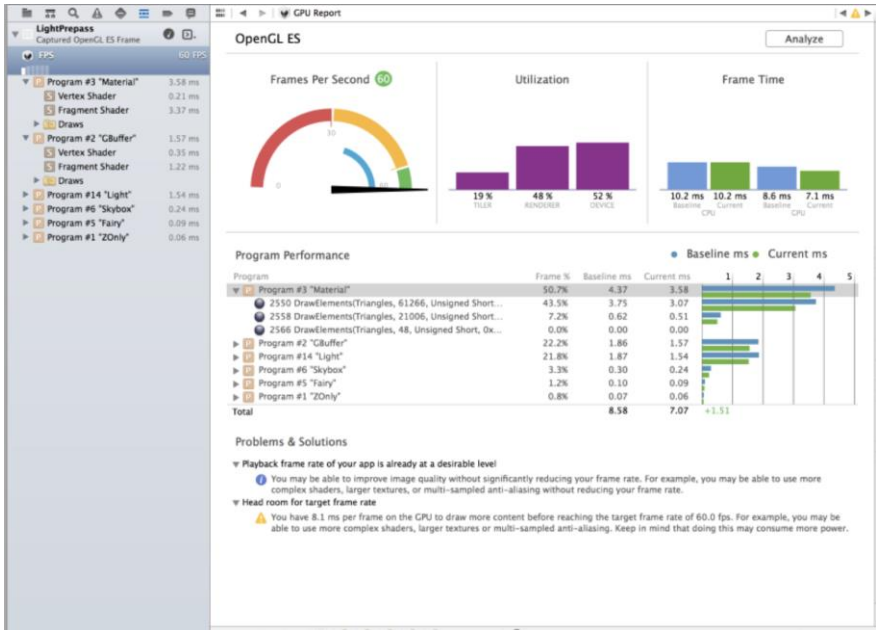


Рисунок 2.3 – Тестування продуктивності роботи додатків в Xcode

Під час тестування відображається навантаження головного процесору на лінії часу. Якщо головний процесор пристрою має декілька ядер або декілька потоків, то

існує можливість переглянути розподілення навантажень. Також у процесі тестування демонструється використання операційної пам’яті. Враховуючи відображані данні розробник може оптимізувати код щоб уникнути зайвих навантажень, зменшити споживання енергоресурсів та уникнути зависань додатку.

Розробнику дається можливість зберігати усі данні отримані під час тестування для подалшого аналізу або порівняння під час нових тестів. Користувач Xcode може зберегти усі отримані данні в окремий файл. Усі збережені данні мають ідентифікатори проектів та збірок, що допомагає зрозуміти на якому етапі розробки було проведено тестування.

Мова програмування Swift – багатопарадигмова компільована мова програмування, розроблена компанією Apple для того, щоб співіснувати з Objective C і бути стійкішою до помилкового коду.

## 2.2 Огляд мов програмування

У сучасному світі процес розвитку стрибнув далеко у перед. Останнім часом майже в кожної людини є мобільний телефон або комп’ютер. Майже будь яка електроніка потребує наявність програми яка буде керувати процесом роботи. Програмування є невід’ємною частиною в створенні сучасних електронних пристроїв. Минуло багато часу з моменту появи перших комп’ютерів, але принцип їх роботи не змінився. Потужності обчислювальних машин зростає, але вони все також розуміють тільки наявність, або відсутність сигналу. Для зручного написання програм були розроблені сучасні мови програмування. Сучасні мови все більш схожі на звичайне спілкування. Вже сьогодні розробник пише код звичайними висловлюваннями та словами.

Компанія Apple почала своє існування з виробництва компьютерів. З часом купертіновці зрозуміли що для свого продукту краще використовувати свою мову програмування та розробили мову Objective-C. Пізніше була анонсована більш

сучасна мова Swift. Ця мова використовується для написання мобільних та десктопних додатках під пристрої компанії Apple.

Мова також пропонує безліч сучасних методів програмування, таких як замикання, узагальнене програмування, лямбда-вирази, кортежі і словникові типи, швидкі операції над колекціями, елементи функційного програмування.

Основним застосуванням Swift є розробка користувацьких застосунків для MacOS X і Apple iOS з використанням фреймворка Cocoa і Cocoa Touch. При цьому Swift надає об'єктну модель, сумісну з Objective-C. Вихідний код мовою Swift може змішуватися з кодом на C і Objective-C в одному проекті.

Мова Swift щільно інтегрований у власницьке середовище розробки Xcode і не може бути використаний відособлено на платформах, відмінних від OS X.

Окремо варто відзначити, що Swift від компанії Apple не варто плутати з досить давно розроблюваною скриптовою мовою Swift, націленої на багатонитеве програмування і поставленого під вільною ліцензією Apache.

Мова Swift-програми компілюються у машинний код, що дозволяє забезпечити високу швидкодію. За заявою Apple, код Swift виконується в 1.3 рази швидше коду на Objective-C. Замість збирача сміття Objective-C в Swift використовуються засоби підрахунку посилань на об'єкти, а також надані у LLVM оптимізації, такі як автовекторизація.

Обрана мова переглядає загальноприйняті умови успадкування. Таким чином, більше не потрібно ставити кому в кінці рядка або писати круглі дужки, щоб оточити умовні вирази всередині операторів if / else [7]. Найбільшою перевагою Swift перед Objective-C є те, що виклики методу не розташовуються усередині один одного. Для виклику функцій і методів в Swift використовується стандартний, розділений комами, список параметрів в круглих дужках. В результаті отримуємо мову зі спрощеним синтаксисом і граматикою.

Мова програмування Swift вважається дуже потужним інструментом та незважаючи на те що була випущена не так давно, дуже швидко набирає популярність завдяки своїй гнучкості та швидкодії (рис. 2.4).

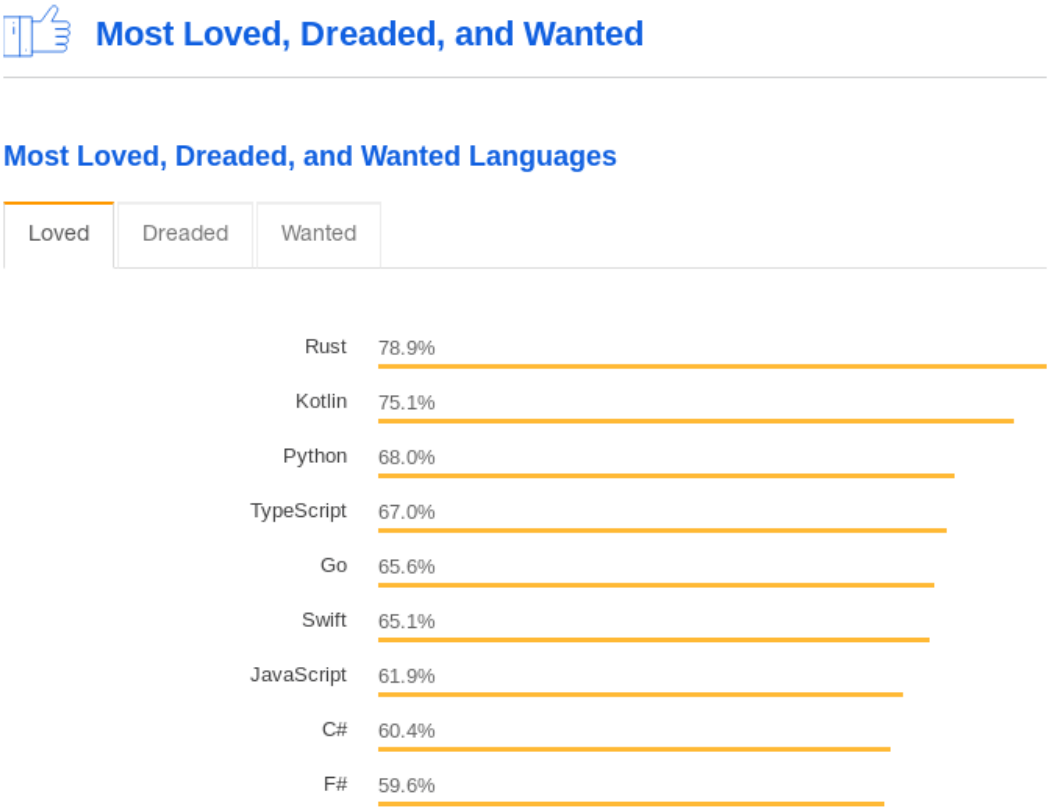


Рисунок 2.4 – Популярність мови програмування Swift серед новачків

Компілятор Swift використовує технологію вільного проекту. Успадкування багатьох елементів мов C та Objective-C дає змогу легко зрозуміти синтаксис та принцип дії. Окрім успадкованих частин, у мові присутні нові засоби керування пам’ятю, тому нова мова є менш вимоглива та більш продуктивна.

В коді також можуть бути присутні опціональні знаки які дають можливість уникати зайвих критичних ситуацій. Опціональні типи допомогають визначити наявність даних, перевірити їх та впевнено застосовувати якщо усе виконано вірно.

Ще одною привігегією є можливість використовувати динамічні бібліотеки. Ці бібліотеки є зовнішніми для використання програмами, але при завантаженні та

встановленні додатку додають себе. Такім чином додаток завантажую код бібліотеки тільки тоді коли він потрібен.

## 2.3 Висновки до розділу

Розробка мобільного додатку є дуже цікавим напрямом та дає багато можливостей у своїй галузі. В даному розділі був проведен аналіз мови програмування Swift. Було доведено та обгрунтовано рішення стосовно вибору мови програмування та середовища для розробки мобільного додатку. Обраний тип роботи є найбільш доступним та зрозумілим для сучасної людини.

# 3 ОСНОВНІ МЕТОДИ ТА ОСОБЛИВОСТІ ТЕХНОЛОГІЙ РОЗРОБКИ

Додаток написаний використовуючи фреймворк UIKit. З цього випливає те що використовуються екземпляри класу UIViewController.

## 3.1 Цикл життя UIViewController

Паттерн проектування додатку Model View Controller (рис. 3.1).

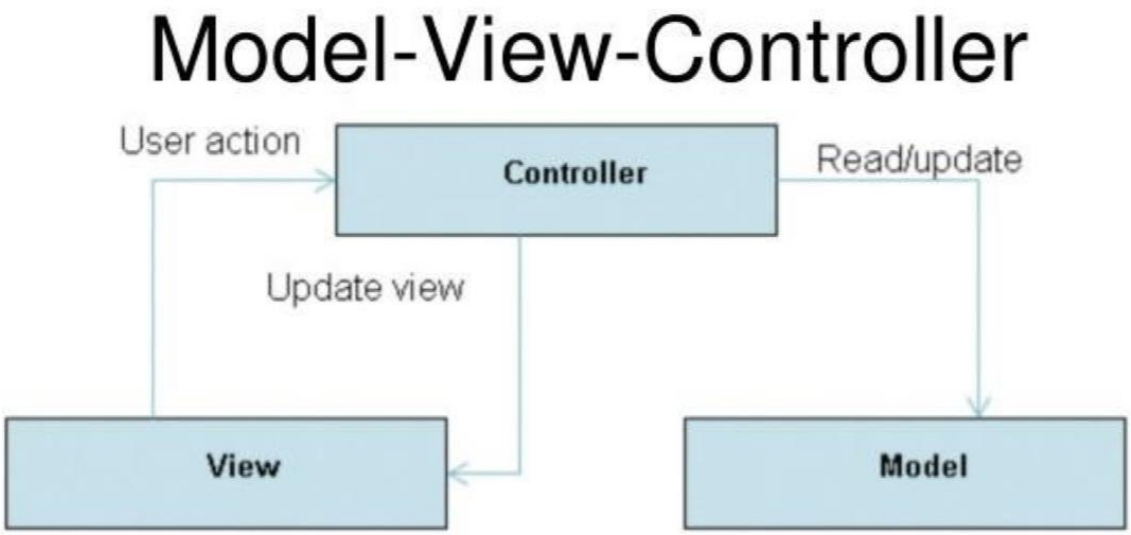


Рисунок 3.1 – Паттер проектування Model View Controller

Життєвий цикл являє собою наступне :

Створення UIViewController:

— Init;

— initWithNibName.

Знищення UIViewController:

— dealloc.

Створення UIView:

— isViewLoaded;

— loadView;

— viewDidLoad initWith.

Обробка зміни стану UIView:

— viewDidLoad;

— viewWillAppear;

— viewDidAppear;

— viewWillDisappear ;

— viewDidDisappear ;

— viewDidUnload.

Знищення UIView:

— viewDidUnload.

Обробка браку пам'яті

— didReceiveMemoryWarning.

На початку життєвого циклю UIViewController виконується ініціалізація, спрацьовує метод init. Нічого окрім самого контроллеру не ініціалізується, та других елементів ще не існує. Виклик методів інших елементів призведе до помилки. Наявність інших елементів перевіряться у методі isViewLoaded.

Екземпляри класу UIView створюються саме перед викликом методу viewDidLoad. Елементи створені самостійно можуть бути ініціалізовані після ініціалізації контроллеру.

Завдяки технології ARC усі вказівники пам’яті на контроллер будуть стерті після його закриття. Розробник може самостійно написати шлях звільнення операційної пам’яті у разі її нестачі, або порядок дій буде виконаний за замовчуванням.

## 3.2 Середа розробки Storyboard

Середа розробки Storyboard має багато переваг. Лише у конструкторі розробник може створити додаток навіть не написавши коду. Переходи між екранами створюються лише за допомогою вказівок між елементами.Кожен елемент доданий у Storyboard може бути налаштований та скріплений з іншими елементами екрану. Усі зв’язки можна побачити та перевірити у середовищі не запускаючи додатку (рис. 3.2).

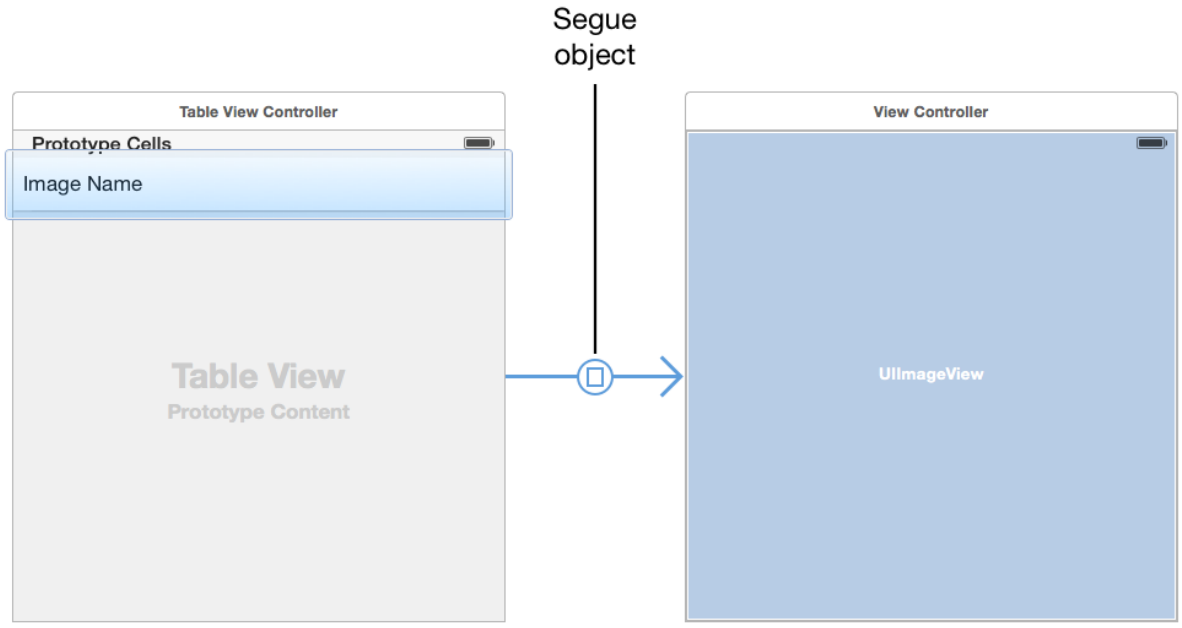


Рисунок 3.2 – Налаштування зв’язків між екранами у Storyboard

Під час розробки додатку була використана технологія AutoLayout. Данна технология дозволяє побудувати динамічні зв’язки між елементами інтерфейсу. Це дозволяє адоптувати додаток до будь яких розмірів екраів. Геометричні зв’язки встановлюються між елементами інтерфейсу за допомогою constrains. Закріплені відстані примусово розміщують елементи на екрані та вказують на можливі зміни при необхідності. Технологія автоматично обчислює необхідні позиції елементів.

Зв’язки можна встановлювати як в Storyboard, так і програмно за допомогою коди створювати та додавати до елементів.

## 3.3 Фреймворк UIKit

Основним фрейворком при створенні додатку мовою Swift є UIKit. Цей фреймворк об’єднує в собі усі елементи інтерфейсу та контролери навігації (рис 3.3).

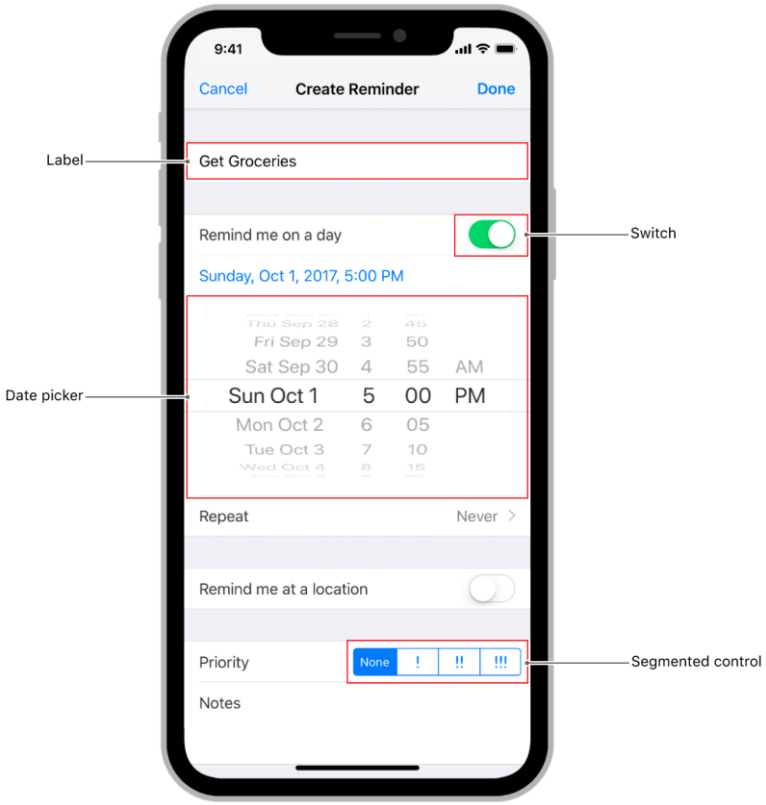


Рисунок 3.3 – Приклад елементів інтерфейсу

Другим за значимістю є фреймворк Foundation. Він використовується тільки при розробці мобільних додатків.

Ключові дії виконує основні задачі необхідні для реалізації таких можливостей як:

— керування додатком;

— керування елементами інтерфейсу;

— підтримка роботи графічної частини та елементів класу UIView;

— забезпечення багатозадачності;

— обробка другу тексту за допомогою віртуальної клавіатури;

— обробка жестів на натискань на елементи інтерфейсу;

— елементи що відображають стандартні вікна та елементи керування;

— підтримка web-інтерфейсу;

— підтримка редагування текстів;

— керування анімацією на екрані;

— забезпечення інтеграції зі сторонніми системами;

— підтримка центру нотіфікацій;

— підтримка елементів для додаткових засобів керування додатком;

— обробка PDF файлів;

— обробка дій елементів зв’язаних з віртуальною клавіатурою.

Одним із основних елементів фреймворку UIKit є клас UITableView. Екземпляри цього класу відображають таблиці які складаються з елементів екземпляру класу UITableViewCell (рис. 3.4).

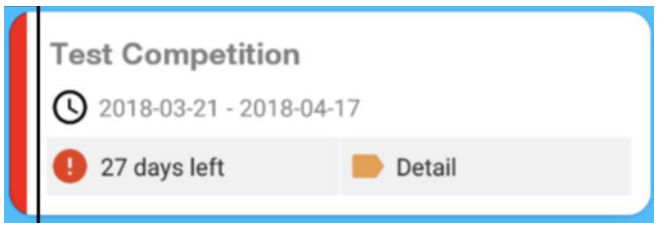


Рисунок 3.4 – екземпляр класу UITableViewCell

Ці елементи можуть розпізнавати натисканння користувача, реагувати на них, та делегуються розробником якщо це необхідно.

## 3.4 Сервіс Firebase

Сервіс Firebase є платформою для мобільних додатків від компанії Google. Цей сервіс дає індивідуальну хмарову базу даних для додатку розробника як сервіс, яка працює в режимі реального часу. З'єднання з базою даних виконується за допомогою API. Розробник має можливість зберігати данні використовуючу сервіс, реалізувати обмін даними між користувачами, та надсилати push повідомлення.

Доступ до бази даних також можна отримати через REST API. Розробники використовуючи бази данних реального часу мають можливість дотримуватися правил безпеки які використовуються на сервері для захисту даних. Сервіс надає безпечний обмін даннними. Розробник може зберігати файли різних форматів та вмісту (рис. 3.5).

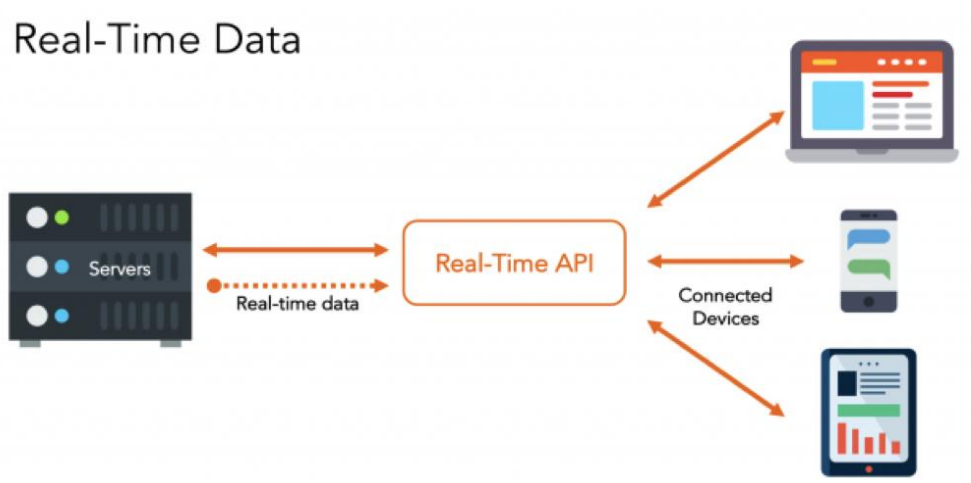


Рисунок 3.5 – Робота додатку з Firebase

Передача данних між мобільним додатком та службою здійснюється через протокол HTTPS.

## 3.5 Висновки до розділу

У розділі було розглянуто використані технології та паттерн проектування. Було обгрунтоване використання фреймворків та наведені приклади їх використання під час розробки. Також було пояснено чому саме сервіс Firebase був обраний для зберігання та обміном інформацією.

# 4 АНАЛІЗ РОБОТИ ВЕНТЕЛЯЦІЇ У ПРИМІЩЕННІ

У сучасному світі суспільство вимушено багато часу проводити у приміщеннях. Це може бути офіс, цех виробництва, торгівельний центр, квартира, або загородний будинок.

## 4.1 Основні причини використання вентеляції у приміщенні

У таких приміщеннях частіше за все використовують приточно відточну систему вентиляції (рис 4.1).

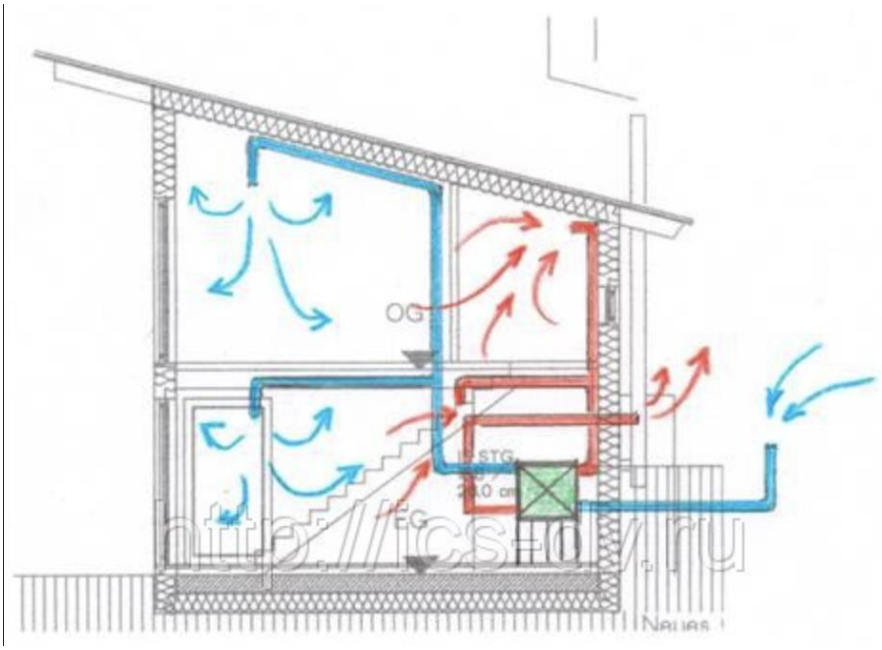


Рисунок 4.1 – Приточно відточна система вентилції

Дихання є необхідною складовою нашого життя. Але якість повітря теж грає важливу роль. Велика кількість вуглекислого газу може призвести до сонливості, швидкого втомлення та паморочення в голові. Такі симптоми можна покачити у людей які постійно знаходяться у закритих офісах з обмеженим обміном повітря.

Такі умови погано впливають не лише на людину, але і на компанію в якій вона працює. Недостатня якість повітря може призвести до повільної роботи працівників та до помилок через неуважність.

Щоб запобігти похудшення самопочуття людини та зменшення обсягів виконаної роботи для компаній необхідно постійно провітрювати приміщення. Частіш за все для рішення цієї проблеми встановлюють системи вентиляції, але вони не завжди можуть впоратися з обсягами та концентрацією неблагодійних складових повітря.

## 4.2 Норми показників повітря у приміщенн

Нормами вуглекислого газу у приміщенні є 800 — 1 000 ppm. Такі показники є допустимими у офісах, школах, квартирах. При підвищенні показників треба провітрювати приміщення. Повітря з показником вище за 1 500ppm вважається низькоякісним.

Гранично допустимим значенням для чадного газу є 20 мг/м3. При більшій концентрації необхідно як найшвидше провітрити приміщення та за можливістю покинути його до часу поки не зменшиться концентрація газу (рис. 4.2).


Рисунок 4.2 – Вплив вуглексислого газу на людину

Вологість повітря не є дуже суттєвим показником та не шкодить здоровью людини при будь яких показниках, але рекомендується дотримувати вологість

повітря у приміщенні більш ніж 40% та меньш ніж 60% для комфортного перебування.

## 4.3 Причини підвищення концентрації газів

Для комфортного перебування людини у приміщенни потрібно підтрімуваті усі ці показники у нормі. Підвіщення концентрації газів може викликати велика кількість людей в одному приміщенні, спалахи на виробництві, або виділення газів під час хімічних процесів під час виробництва.

Приміщення з дизельними генераторами, або іншими джерелами викиду газів мають бути осшащенні системою вентиляції. У подібних місцях концентрація шкідливих речовин може підвищитися дуже швидко та не завжди може бути помітною для персоналу. Подібний сценарій призводить до поганих наслідків. Створені умови викликають похудшення здоровья людей перебуваючих на території.

## 4.4 Засоби запобігання погіршення якості повітря

Для запобігання погіршення якості повітря в офісі потрібно встановлюваті системі вентиляції розраховані під площу приміщень. Персонал повинен мати достатню площу для особистого користування та попереджувати руководство я разі невиконання норм робочої зони. Температура у приміщенні повинна задовільняти нормам та контролюватися персоналом перебуваючим в офісі.

На виробництві підтримка норм якості повітря є дуже важливим фактором. Концентрація шкідливих речовин на виробніцтві дуже часто може не відповідаті нормам. У разі відхилень від норм потрібно увімкнути вентиляцію або покинути приміщення до стабілізації показніків. На підпріємствах потрібно щоб була людина

яка відповідає за контроль якості повітря оскільки це впливає як на здоров’я персоналу та може викликати надзвичайний стан

## 4.5 Висновки до розділу

У розділі був проведен аналіз основних складових повітря та наведені обгрунтована необхідність використання вентиляції у приміщеннях. Були наведені гранично допустимі значення показників для комфортного перебування людини у приміщенні без загрозу життю та здоров’ю.

# 5 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Перед кожним розробником завжди постає питання: які саме інструменти вибрати для роботи. Головними критеріями стають простота у використанні та швидкість.

## 5.1 Структура програми

Від простоти використання та швидкості залежить як якість написаного коду так і вподобання користувачам.

Порівнюючи Swift і Objective-C можна з впевненістю сказати, що Swift – це сучасні норми синтаксису, ефективне управління пам'яттю, висока швидкість роботи і інтерактивність (рис 5.1).

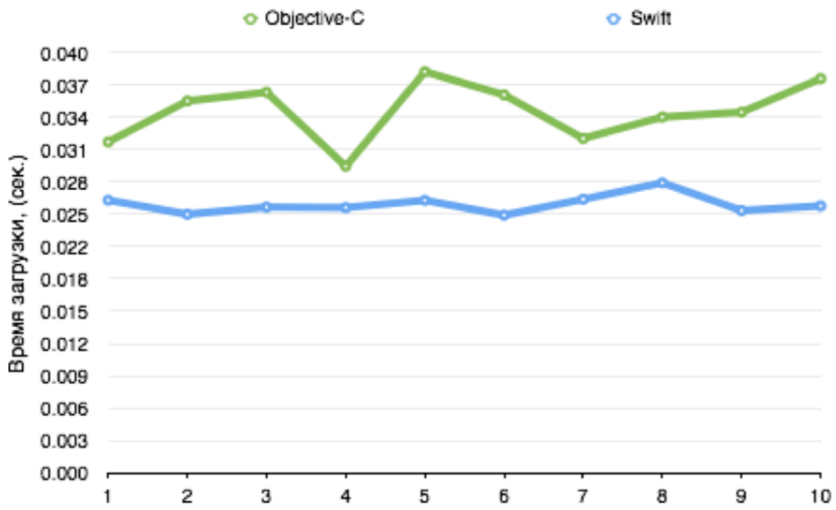


Рисунок 5.1 – Робота додатку з Firebase

У Objective-C цього немає, але зате є надійність, база документації, прикладів, шаблонів і багато досвідчених програмістів. Swift – це майбутнє, яке вже не за

горами. Все пізнається в порівнянні і лише відпрацювавши певну кількість проектів, можна стверджувати, чи зручна мова, і чи можна на ній ефективно працювати.

Проаналізувавши особливості кожного з середовищ розробки, Xcode перемагає усіх вищепредставлених учасників, адже працювати в ньому – одне задоволення. Воно гнучке, швидке, потужне і здатне завжди прийти на допомогу. Xcode стає все краще, незважаючи на складні заходи, які вживаються Apple з метою утримання повного контролю над iOS додатками і пристроями (рис 5.2).

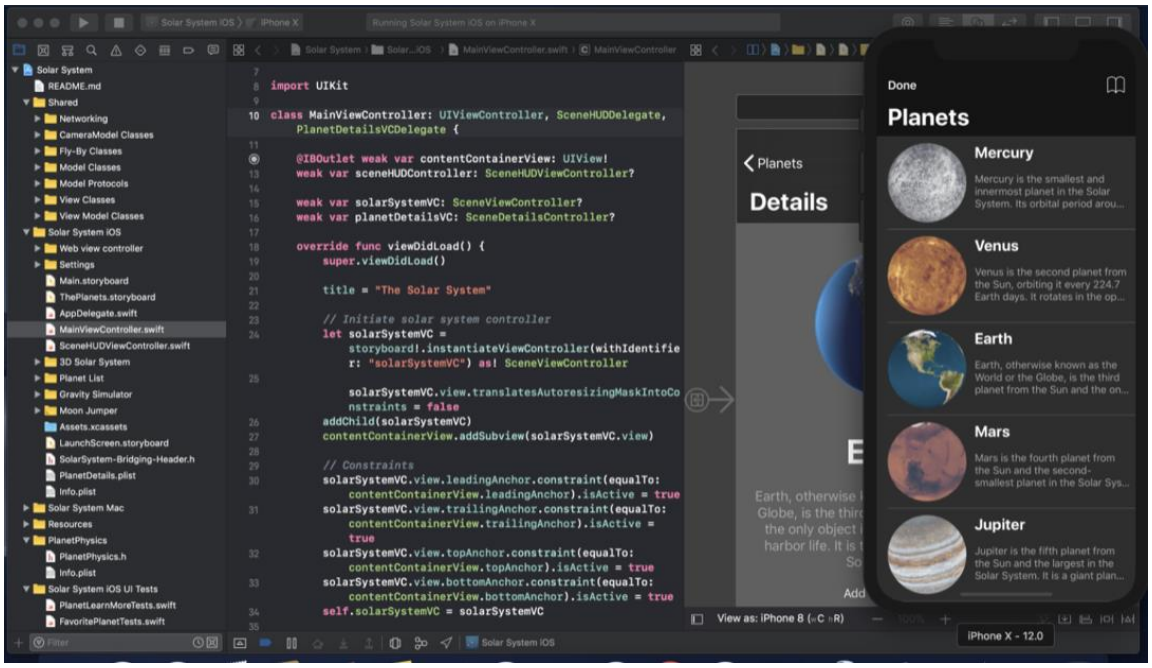


Рисунок 5.2 – Робота додатку з Firebase

Робочий простір в Xcode тримає розробника зосередженим. Під час написання коду, в реальному часі можна побачити помилки компілятора або проблеми, а також повідомлення з докладним описом корисної інформації. Відладчик працює плавно, а симулятор – швидкий і орієнтований на користувача. Отже, Xcode перевершує всі інші IDE.

Для реалізації задачі управління та моніторингу вентеляційної системи був розроблений мобільний додаток. Додаток має підключення до сервісу Firebase для

зберігання та обміну інформацією. Застосунок отримує інформацію з серверу стосовно стану повітря на вулиці, отримана інформація аналізується та порівнюється з нормами складу повітря. За кожним показником виставляється індикатор границь концентрації. Обробивши усі показники додаток створює показник що показує загальну якість повітря. Отримавши загальну якість повітря додаток відображає користувачу рекомендації з використання вентиляційної системи (рис 5.3).

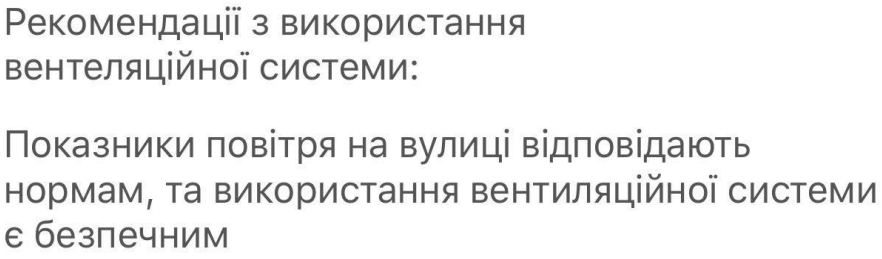


Рисунок 5.3 – Скріншот рекомендацій додатку стосовно використання вентиляційної системи

Користувач також бачить усі показники повітря на вулиці самостійно. Аналізуючи потік повітря система виводить користувачу індикатор стану фільтру для своєчасної заміни (рис. 5.4).


Рисунок 5.4 – індикатор стану фільтру

Після аналізу показників повітря на вулиці, якщо використання системи не вплине негативно на повітря у приміщенні, система починає аналізувати повітря у

кожному приміщенні. Аналогічно з аналізом повітря на вулиці аналізується повітря у кожному приміщенні (рис. 5.5).

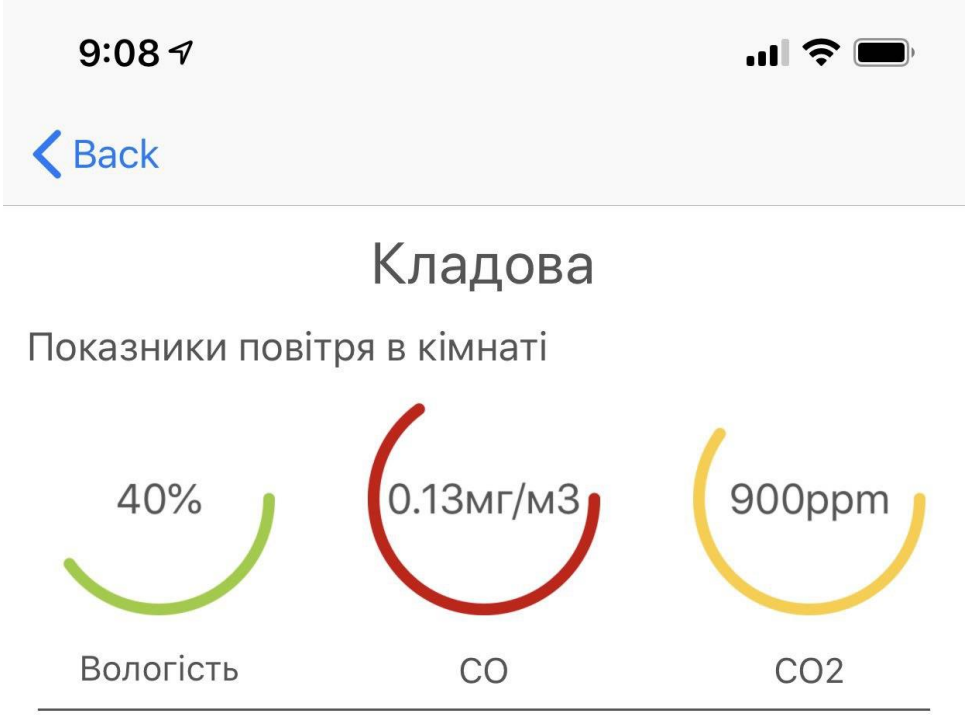


Рисунок 5.4 – Індикатори показників повітря у приміщенні

Виставивши індикатори стану повітря у кожній кімнаті, додаток визначає коефіцієнти необхідності вентиляції та розставляє пріоритети (рис. 5.5).


Рисунок 5.5 – Розподілення потужності по кімнатам

Розставивши пріорітети по усім приміщенням система розподіляє потужність враховуючи отримані данні. Якщо показники повітря в усіх приміщеннях

відповідають нормам, система буде у режимі очікування та не буде працювати без необхідності.

Структура проекту побудована так, щоб кожен модуль відповідав за свої окремі задачі та виконував їх за необхідністю. До проекту під’єднан фреймворк Firebase який включає в себе класи та методи для використанння та зв’язку з сервісом від компанії Google. Файли проекту розподілені та названі відповідно своїм задачам (рис. 5.6).

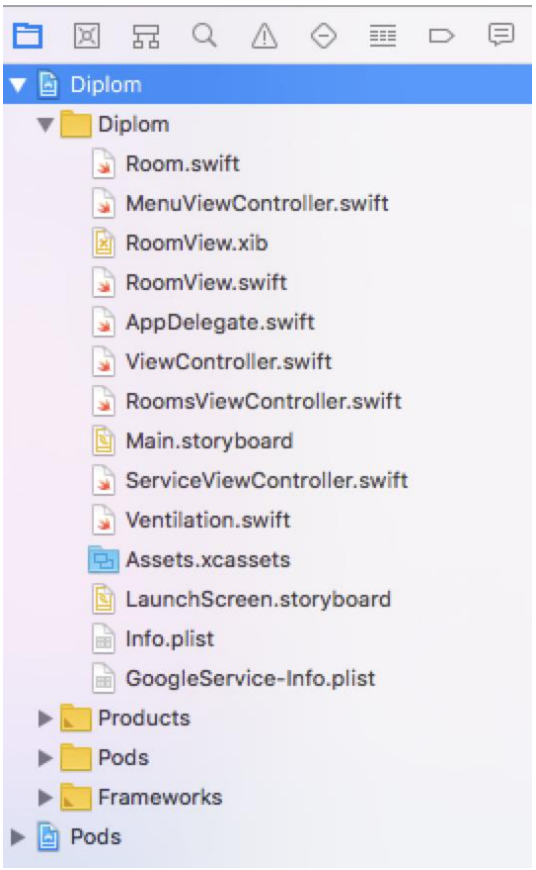


Рисунок 5.6 – Робота додатку з Firebase

Модуль Pods складається з класів, що забезпечують під’єднання та роботу з Firebase, для роботи якого потрібне підключення до мережі.

Модуль Diplom являє собою набір класів на мові програмування Swift. У цьому модулі реалізована уся бізнес логіка додатку та класи що використовуються

для обміну данними з Firebase. Включає в себе також файл AppDelegate що керує додатком під час усього життєвого циклу. Файл Main.storyboard - дизайн екранів додатку та зв’язки між ними. Файл info.plist зберегає у собі усі налаштування додатку та параметри надання доступу для роботи зі сторонніми сервісами. Модуль Assets містить у собі усі графічні елементи які застосовуються під час роботи застосунку та відображаються в інтерфейсі.

## 5.2 Опис існуючих програмних рішень

Програмний агент моніторингу та управління вітроенергетичної установки - технічна система, що забезпечує процес управління вентиляційною системою, встановлення розкладу для роботи системи, примусове керування вентиляцією у ручному режимі.

Наразі існує багато систем, що забезпечують роботу вентиляції у приміщеннях за встановленим розкладом, або за встановленим інтервалом роботи.

Прикладом такої системи є мобільний додаток “multiMATIC”. Подібні системи не враховують стан повітря та не слідкують за повітрям на вулиці. Такі системи не завжди забезпечують необхідний обмін повітря у приміщенні та не є енергоефективними. Системи можуть працювати тоді коли це не є необхідним, або навпаки не працювати тоді коли це є необхідним. Також увімкнення таких систем може бути небезпечним у разі небезпечного складу повітря на вулиці, тому що це не враховується при увімкненні та користувач не отримує попереджень

## 5.3 Користувацький інтерфейс та можливості застосунку

Для автоматизації процесу моніторингу та управління вентиляційною установкою необхідна система, яка буде надавати найважливішу інформацію для

користувача та давати можливість контролювати весь процес вентиляції у приміщенні.

Вибір мобільної архітектури обумовлений тим що архітектура iOS схожа з базовою архітектурою операційної системи Mac OS X. На найвищому рівні iOS являє собою проміжний шар між обладнанням пристроя та додатками, які відображаються на екрані.

На головному екрані мобільного додатку відображаються показники повітря на вулиці, рекомендації стосовно використання вентеляційної системи та індикатор стану фільтру (рис. 5.7).


Рисунок 5.7 – Головний екран додатку

З головного екрану користувач може перейти до екрану керування та стану вентиляційної системи. На екрані стану та керування відображаються усі

приміщення підключені до системи. Для кожного приміщення виведені індикатори показників повітря які відстежують відповідність нормам.

Нижче присутні індикатори стану вентиляційної системи та обраного режиму роботи. Присутні елементи примусового керування системою, щоб увімкнути вентиляцію у обраному приміщенні за необхідністю. При відображені кожної кімнати виводиться інформація про поточну потужність виділену системою для кожної кімнати (рис. 5.8).


Рисунок 5.8 – Екран стану та керування вентиляційною системою

Усі індикатори змінюють кольори залежно від показників для більш зрозумілого відображення інформації стосовно повітря у приміщеннях.

## 5.4 Висновки до розділу

У цьому розділі розроблено мобільний додаток що відповідаю усім поставленим задачам, є простим в використанні та зрозумілим користувачеві. У створеному додатку були реалізовані функції яких не вистачає в існуючих програмних рішеннях. Додатком можна користуватися та керувати системою не знаходячись поряд з нею.

# ВИСНОВКИ

В процесі виконання роботи було виконано наступне:

— отримано необхідні знання для створення системи обліку та оперативного зберігання проектно-технологічної документації;

— сформульована проблематика розробки програмного агента автоматизації моніторингу та управління вентиляційної системи. Для вирішення поставленої задачі було вирішено створити мобільний додаток з хмаровою базою даних, адже завдяки такому підходу користувачі мають доступ до системи з мобільного телефону без необхідності находитись поруч з самою системою вентиляції;

— Розроблено систему що надає користувачу вибір режиму роботи, забезпечує своєчасне постачання свіжого повітря та контролює якість повітря. Система аналізує показники отримані з датчиків встановлених у приміщенні, перевіряє отримані данні з нормами та розподіляє потужність по кімнатам в яких потребується вентиляція. Увесь моніторинг та керування здійснюється у мобільному додатку та реалізована на мові програмування високого рівня Swift.

# ПЕРЕЛІК ПОСИЛАНЬ

1. Офіційний сайт Apple Xcode. – Режим https://developer.apple.com/xcode/. – Дата доступу 25.05.2019.
2. Офіційний сайт AppCode. – Режим https://www.jetbrains.com/objc/. – Дата доступу 21.05.2019.
3. Офіційний сайт Xamarin. – Режим доступу: https://www.xamarin.com/. – Дата доступу 19.05.2019.
4. Офіційний сайт Visual Studio. – Режим https://www.visualstudio.com/. – Дата доступу 03.05.2019.
5. Офіційний сайт Appcelerator. – Режим http://www.appcelerator.com/. – Дата доступу 02.06.2019.
6. ANON The Swift Programming Language (Swift 2.1) / ANON – Cupertino: Apple Inc., 2014. – 528 p.
7. Vandad Nahavandipoor iOS 8 Swift Programming Cookbook / Vandad Nahavandipoor – Boston: O'Reilly Media., 2014. – 902 p.
8. Офіційна документація мови програмування Swift. – Режим доступу: https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swi ft_Programming_Language/. – Дата доступу 12.05.2019.
9. iOS 12 Programming Fundamentals with Swift – Boston: O'Reilly Media., 2018. – 646 p.
10. Learning Swift: Building Appls for macOS, iOS, and Beyond - Boston: O'Reilly Media., 2018. – 378 p.
11. Beginner’s Guide to iOS 11 App Development Using Swift 4: Xcode, Swift and App Design Fundamentals - CreateSpace Independent Publishing Platform; 2017. 216 p.
12. iOS 12 Programming for Beginners: An introductory guide to iOS app development with Swift 4.2 and Xcode 10, 3rd Edition – Packt Publishing., 2018. – 692 p.

13. iOS Apps for Masterminds 4th Edition: How to take advantage of Swift 4.2, iOS 12, and Xcode 10 to create insanely great apps for iPhone and iPads – CreateSpace Independent Publishing Platform., 2018. – 376 p.
14. Advanced iOS Architecture (First Edition): Real-world app architecture in Swift – Razeware LLC., 2019. – 307 p.
15. Swift Programming: The Big Nerd Ranch Guide (3rd Edition) - Big Nerd Ranch Guides., 2019. – 480 p.

Додаток 1

# Розробка програмного агента моніторингу та управління системи вентиля

Специфікація

УКР.НТУУ“КПІ”.ТМ51116_19Б

Аркушів 2

2019

| Позначення | Найменування | Примітки |
|---|---|---|
| Документація | | |
| УКР.НТУУ«КПІ ім. Ігоря Сікорського».ТМ51116_19Б 81-1 | Записка | Пояснювальна записка |
| Компоненти | | |
| УКР.НТУУ«КПІ». ТМ51116_19Б 12-1 | Текст програмного модулю | |
| УКР.НТУУ«КПІ». ТМ51116_19Б 13-1 | Опис програми | |

Додаток 2

Розробка програмного агента моніторингу та управління системи вентиля

Текст програмного модулю

УКР.НТУУ“КПІ”.ТМ51116_19Б 12-1

Аркушів 9

2019

```
< //
//  RoomsViewController.swift
//  Diplom
//
//  Created by Mykola Rudenko on 5/9/19.
//  Copyright © 2019 Mykola Rudenko. All rights reserved.
//

import UIKit

let needvar = String()

class RoomsViewController: UIViewController, UIScrollViewDelegate {

    @IBOutlet weak var scrollView: UIScrollView!
    @IBOutlet weak var pageControl: UIPageControl!
    var rooms = [Room]()
    var slides:[RoomView] = []

    override func viewDidLoad() {
        super.viewDidLoad()
//        Singleton.shared.ventilation.autoVent()
        rooms = Singleton.shared.rooms
        scrollView.delegate = self
        slides = createSlides()
        setupSlideScrollView(slides: slides)
        pageControl.numberOfPages = slides.count
        pageControl.currentPage = 0
        view.bringSubviewToFront(pageControl)
        NotificationCenter.default.addObserver(self, selector: #selector(needUpdate),
name: Notification.Name(rawValue: needvar), object: nil)
        print("need room in Singletone \(Singleton.shared.priorityRoomTittle)")
        scrollView.contentSize = CGSize(width: scrollView.contentSize.width,height:
0)



        // Do any additional setup after loading the view.
    }
    @objc func needUpdate()
    {
        print("need UPDATE")
        slides = createSlides()
        setupSlideScrollView(slides: slides)
        scrollView.contentSize = CGSize(width: scrollView.contentSize.width,height:
0)
        print("need room in Singletone \(Singleton.shared.priorityRoomTittle)")
    }
    func requestData(completion: ((_ data: String) -> Void)) {
        rooms = Singleton.shared.rooms
        scrollView.delegate = self
        slides = createSlides()
        setupSlideScrollView(slides: slides)
        pageControl.numberOfPages = slides.count
        pageControl.currentPage = 0
        view.bringSubviewToFront(pageControl)
    }
```

```
    func createSlides() -> [RoomView] {
        var views = [RoomView]()
        rooms.forEach{
            let newRoomView = Bundle.main.loadNibNamed("RoomView", owner: self,
options: nil)?.first as! RoomView
//            newRoomView.airHumodityIndicator.value = CGFloat($0.airHumidity)
//            newRoomView.CO2Indicator.value = CGFloat($0.CO2)
//            newRoomView.COIndicator.value = CGFloat($0.CO)
            if $0.ventPower > 0 {
                newRoomView.statusLabel.text = "Статус: ВКЛ"} else {
                newRoomView.statusLabel.text = "Статус: ВЫКЛ"
            }
            if Singleton.shared.priorityRoomTittle.isEmpty {
                newRoomView.modeLabel.text = "Режим: АВТО"
            } else {
                newRoomView.modeLabel.text = "Режим: РУЧНОЙ"
            }
            newRoomView.tittleLabel.text = $0.tittle
            newRoomView.setCO(needToSet: $0.CO)
            newRoomView.setCO2(needToSet: $0.CO2)
            newRoomView.setairHumidity(needToSet: $0.airHumidity)
            newRoomView.ventPower = $0.ventPower
            newRoomView.setFunPower()
            views.append(newRoomView)
        }
        return views

    }

    func reload() {
        scrollView.reloadInputViews()
    }

    func setupSlideScrollView(slides : [RoomView]) {
        scrollView.frame = CGRect(x: 0, y: 0, width: view.frame.width, height:
view.frame.height)
        scrollView.contentSize = CGSize(width: view.frame.width *
CGFloat(slides.count), height: view.frame.height)
        scrollView.isPagingEnabled = true

        for i in 0 ..< slides.count {
            slides[i].frame = CGRect(x: view.frame.width * CGFloat(i), y: 0, width:
view.frame.width, height: view.frame.height)
            scrollView.addSubview(slides[i])
        }
    }

    func scrollViewDidScroll(_ scrollView: UIScrollView) {
        let pageIndex = round(scrollView.contentOffset.x/view.frame.width)
        pageControl.currentPage = Int(pageIndex)

        let maximumHorizontalOffset: CGFloat = scrollView.contentSize.width -
scrollView.frame.width
        let currentHorizontalOffset: CGFloat = scrollView.contentOffset.x

        // vertical
        let maximumVerticalOffset: CGFloat = scrollView.contentSize.height -
scrollView.frame.height
        let currentVerticalOffset: CGFloat = scrollView.contentOffset.y
```

```
        let percentageHorizontalOffset: CGFloat = currentHorizontalOffset /
maximumHorizontalOffset
        let percentageVerticalOffset: CGFloat = currentVerticalOffset /
maximumVerticalOffset

        let percentOffset: CGPoint = CGPoint(x: percentageHorizontalOffset, y:
percentageVerticalOffset)
    }

}

//
//  Ventilation.swift
//  Diplom
//
//  Created by Mykola Rudenko on 5/10/19.
//  Copyright © 2019 Mykola Rudenko. All rights reserved.
//

import UIKit

class Ventilation: NSObject {

    var isAutoMode = true
    var ventPower = 3
    var havePower = 3
//    var ventilationRooms: [Int:Int] = [Int:Int]()
    var sortedRooms = [Room]()

    func checkRooms() {
        let rooms = Singleton.shared.rooms
        if !rooms.isEmpty {
            for i in 0...rooms.count-1 {
                rooms[i].checkTotalDanger()
            }
            sortedRooms = sortByDanger(arr: rooms)
        }
    }
    func autoVent() {
        print("auto mode func")

        isAutoMode = true
        Singleton.shared.priorityRoomTittle = ""
        checkRooms()
        if !sortedRooms.isEmpty {
            print(sortedRooms)

//            ventilationRooms = [:]
            for i in 0...sortedRooms.count - 1 {
                sortedRooms[i].ventPower = 0
                }
            havePower = 3
            for i in 0...sortedRooms.count - 1 {
                print("total danger for \(sortedRooms[i].tittle) is
\(sortedRooms[i].totalDanger)")
                sortedRooms[i].ventPower = selectPower(needPower:
sortedRooms[i].totalDanger)
                print("vent power for \(sortedRooms[i].tittle)
\(sortedRooms[i].ventPower)")
```

```
                print("have power: \(havePower)")

            }
            Singleton.shared.setRooms(newArr: sortedRooms)
            print(Singleton.shared.rooms)
//            }
        }
        NotificationCenter.default.post(name: Notification.Name(rawValue: needvar),
object: self)
    }

    func selectPower(needPower:Int) -> Int {
        switch havePower {
        case 3:
            if needPower == 3 {
                havePower -= 3
                return 3
            }
            if needPower == 2 {
                havePower -= 2
                return 2
            }
            if needPower == 1 {
                havePower -= 1
                return 1
            } else {
                return 0
            }
        case 2:
            if needPower == 3 {
                havePower -= 2
                return 2
            }
            if needPower == 2 {
                havePower -= 2
                return 2
            }
            if needPower == 1 {
                havePower -= 1
                return 1
            } else {
                return 0
            }
        case 1:
            if needPower == 3 {
                havePower -= 1
                return 1
            }
            if needPower == 2 {
                havePower -= 1
                return 1
            }
            if needPower == 1 {
                havePower -= 1
                return 1
            } else {
                return 0
            }
        default:
            return 0
```

```
        }
    }

    func manualMode(tittleNeedRoom: String){
        isAutoMode = false
        turnOffAllVent()
        Singleton.shared.priorityRoomTittle = tittleNeedRoom
        if !sortedRooms.isEmpty {
            for i in 0...sortedRooms.count - 1 {
                if sortedRooms[i].tittle == tittleNeedRoom {
                    sortedRooms[i].ventPower = 3
                }
            }
        }
        Singleton.shared.setRooms(newArr: sortedRooms)
        NotificationCenter.default.post(name: Notification.Name(rawValue: needvar),
object: self)

    }

    func turnOffAllVent() {
        checkRooms()
        if !sortedRooms.isEmpty {
            for i in 0...sortedRooms.count - 1 {
                sortedRooms[i].ventPower = 0
            }
        }
    }

    func sortByDanger(arr:[Room]) -> [Room] {
        if !arr.isEmpty {
            var newArr = [Room]()
            for i in 0...arr.count-1 {
                if arr[i].totalDanger == 3 {
                    newArr.append(arr[i])
                }
            }
            for i in 0...arr.count-1 {
                if arr[i].totalDanger == 2 {
                    newArr.append(arr[i])
                }
            }
            for i in 0...arr.count-1 {
                if arr[i].totalDanger == 1 {
                    newArr.append(arr[i])
                }
            }
            for i in 0...arr.count-1 {
                if arr[i].totalDanger == 0 {
                    newArr.append(arr[i])
                }
            }
            return newArr
        } else {
            return []
        }
    }

}
```

```
//
//  MenuViewController.swift
//  Diplom
//
//  Created by Mykola Rudenko on 5/10/19.
//  Copyright © 2019 Mykola Rudenko. All rights reserved.
//

import UIKit
import UICircularProgressRing

class MenuViewController: UIViewController {

    @IBOutlet weak var CO2Indicator: UICircularProgressRingView!
    @IBOutlet weak var COIndicator: UICircularProgressRingView!
    @IBOutlet weak var airHumidityIndicator: UICircularProgressRingView!
    override func viewDidLoad() {
        super.viewDidLoad()
        Singleton.shared.outside.checkTotalDanger()
        setCO(needToSet: Singleton.shared.outside.CO)
        setCO2(needToSet: Singleton.shared.outside.CO2)
        setairHumidity(needToSet: Singleton.shared.outside.airHumidity)
        let room1 = Room(tittle: "room1", CO2: 600, CO: 0.09, O: 60, airHumidity: 20,
isVentilationOn: true)
        let room2 = Room(tittle: "room2", CO2: 900, CO: 0.13, O: 46, airHumidity: 40,
isVentilationOn: false)
        let room3 = Room(tittle: "room3", CO2: 1000, CO: 0.14, O: 32, airHumidity:
78, isVentilationOn: false)
        let room4 = Room(tittle: "room4", CO2: 1500, CO: 0.06, O: 870, airHumidity:
32, isVentilationOn: false)
        if Singleton.shared.rooms.isEmpty {
            Singleton.shared.rooms = [room1,room2,room3,room4]
            Singleton.shared.ventilation.autoVent()
        }


        // Do any additional setup after loading the view.
    }

    func setCO(needToSet:Double){
        COIndicator.value = CGFloat(needToSet)
        if needToSet <= 0.03 {
            COIndicator.innerRingColor = UIColor.init(red: 153/255, green: 204/255,
blue: 51/255, alpha: 1)
        } else if (needToSet > 0.03 && needToSet <= 0.08) {
            COIndicator.innerRingColor = UIColor.init(red: 255/255, green: 204/255,
blue: 51/255, alpha: 1)
        } else if (needToSet > 0.08 && needToSet <= 0.12) {
            COIndicator.innerRingColor = UIColor.init(red: 255/255, green: 102/255,
blue: 0/255, alpha: 1)
        } else if (needToSet > 0.12) {
            COIndicator.innerRingColor = UIColor.init(red: 204/255, green: 0/255,
blue: 0/255, alpha: 1)
        }
    }

    func setCO2(needToSet:Double) {
        CO2Indicator.value = CGFloat(needToSet)
        if needToSet <= 800 {
```

```
            CO2Indicator.innerRingColor = UIColor.init(red: 153/255, green: 204/255,
blue: 51/255, alpha: 1)
        } else if (needToSet > 800 && needToSet <= 1000) {
            CO2Indicator.innerRingColor = UIColor.init(red: 255/255, green: 204/255,
blue: 51/255, alpha: 1)
        } else if (needToSet > 1000 && needToSet <= 1400) {
            CO2Indicator.innerRingColor = UIColor.init(red: 255/255, green: 102/255,
blue: 0/255, alpha: 1)
        } else if (needToSet > 1400) {
            CO2Indicator.innerRingColor = UIColor.init(red: 204/255, green: 0/255,
blue: 0/255, alpha: 1)
        }
    }

    func setairHumidity(needToSet:Double) {
        airHumidityIndicator.value = CGFloat(needToSet)
        if needToSet < 30 && needToSet > 60 {
            airHumidityIndicator.innerRingColor = UIColor.init(red: 255/255, green:
204/255, blue: 51/255, alpha: 1)
        } else {
            CO2Indicator.innerRingColor = UIColor.init(red: 153/255, green: 204/255,
blue: 51/255, alpha: 1)
        }

    }
    /*
    // MARK: - Navigation

    // In a storyboard-based application, you will often want to do a little
preparation before navigation
    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        // Get the new view controller using segue.destination.
        // Pass the selected object to the new view controller.
    }
    */

}

//
//  Room.swift
//  Diplom
//
//  Created by Mykola Rudenko on 4/7/19.
//  Copyright © 2019 Mykola Rudenko. All rights reserved.
//

import UIKit

class Room: NSObject {

    var tittle = String()
    var totalDanger = Int()
    var ventPower = 0
    var afterFilterPower = Int()

    var CO2 = Double(){
        didSet{
            self.checkCO2DangerLevel()
        }
    }
```

```
    var CO2DangerLevel = Int()
    var CO = Double(){
        didSet{
            self.checkCODangerLevel()
        }
    }
    var CODangerLEvel = Int()
    var O = Double(){
        didSet{
            self.checkODangerLevel()
        }
    }
    var ODangerLevel = Int()
    var airHumidity = Double(){
        didSet{
            self.checkAirHumidityDangerLevel()
        }
    }
    var airHumidityDangerLevel = Int()
    var isVentilationOn = Bool()

    func checkAir() {

    }

    func checkNoDanger() -> Int{
        var dangerLevel = 0
        if(CO2 >= 0.06 && CO2 <= 0.08) && CO <= 0.0024 && (airHumidity >= 30 &&
airHumidity <= 60) && (O >= 19 && O <= 23) {
            dangerLevel = 0
        }
        return dangerLevel
    }

   func checkCO2DangerLevel(){

    if (CO2 <= 800) {
        CO2DangerLevel = 0
    } else if (CO2 > 800 && CO2 <= 1000) {
        CO2DangerLevel = 1
    } else if (CO2 > 1000 && CO2 <= 1400) {
        CO2DangerLevel = 2
    } else if (CO2 > 1400) {
        CO2DangerLevel = 3
    }
    }

    func checkCODangerLevel(){

        if (CO <= 0.03) {
            CODangerLEvel = 0
        } else if (CO > 0.03 && CO <= 0.08) {
            CODangerLEvel = 1
        } else if (CO > 0.08 && CO <= 0.12) {
            CODangerLEvel = 2
        }else if ( CO > 0.12 ) {
            CO2DangerLevel = 3
        }
    }
```

```
    func checkODangerLevel(){
        ODangerLevel = 0
//        if (CO2 <= 0.08) {
//            CO2DangerLevel = 0
//        } else if (CO2 > 0.08 && CO2 <= 0.1) {
//            CO2DangerLevel = 1
//        } else if (CO2 > 0.1 && CO2 <= 0.11) {
//            CO2DangerLevel = 2
//        } else if (CO2 > 0.11 && CO2 <= 0.13) {
//            CO2DangerLevel = 3
//        } else if (CO2 > 0.13) {
//            CO2DangerLevel = 4
//        }
    }

    func checkAirHumidityDangerLevel(){
        airHumidityDangerLevel = 0
        if (airHumidity > 30 && airHumidity < 60){
            airHumidityDangerLevel = 0
        } else {
            airHumidityDangerLevel = 1
        }
    }



    init(tittle:String, CO2:Double, CO:Double, O:Double, airHumidity: Double,
isVentilationOn:Bool) {
        self.tittle = tittle
        self.CO2 = CO2
        self.CO = CO
        self.O = O
        self.airHumidity = airHumidity
        self.isVentilationOn = isVentilationOn
    }

    func checkTotalDanger(){
        var dangers = [Int]()
        checkCODangerLevel()
        checkCO2DangerLevel()
        checkAirHumidityDangerLevel()
        dangers.append(CODangerLEvel)
        dangers.append(CO2DangerLevel)
        dangers.append(airHumidityDangerLevel)
        totalDanger = (dangers.sorted{$1<$0}).first!
    }

}

class Singleton {

    // MARK: - Shared

    static let shared = Singleton()

    var ventPower = 100
    var rooms = [Room]()
    var priorityRoomTittle = ""
    var ventilation = Ventilation()
```

```
    var outside = Room(tittle: "Outside", CO2: 200, CO: 0.01, O: 80, airHumidity: 31,
isVentilationOn: false)

    func setRooms(newArr: [Room]){
        rooms = newArr.sorted(by: {$0.tittle > $1.tittle})
    }

    private init() {
    }
```

Додаток 3

Розробка програмного агента моніторингу та управління системи вентиля

Опис програмного модулю

УКР.НТУУ“КПІ”.ТМ51116_19Б 13-1

Аркушів 6

2019

# АНОТАЦІЯ

Розроблений програмний продукт реалізує можливість ведення обліку земельних ресурсів.

Користувачами даної системи можуть бути працівники екологічних служб, екологи, працівники зі збору статистики. В загальному програмний продукт призначений для відомчих та інформативних цілей, але, завдяки гнучкості обраних технологій, може бути розширений для використання в реальних умовах праці.

# ЗМІСТ

# 1 ВІДОМОСТІ ПРО ПРОГРАМНИЙ МОДУЛЬ

Даний програмний модуль розроблено у середовищі редактора Xcode, використовуючи мову програмування Swift; сервіс Firebase та деякі додаткові бібліотеки.

Програма призначена для моніторингу складу повітря та автоматичного керування системою вентиляції.

## 1.1. Опис логічної структури

Програмний продукт було розроблено у вигляді мобільного додатку.

Даний мобільний додаток складається з 2 основних екранів —екран моніторингу повітря на вулиці та рекомендацій, та екран стану вентеляційної системи та керування.

До екрану моніторингу повітря на вулиці та рекомендацій входять:

— Індикатори показників складу повітря на вулиці;

— Індикатор стану фільтру;

— Рекомендації з використання вентиляційої системи.

Серед функцій екрану стану вентиляційної системи та керування можна виділити такі як:

— Відображення індикаторів складу повітря у кожій кімнаті;

— Примусове керування системою;

— Відображення поточного розподілення потужності системи.

## 1.2. Вхідні та вихідні дані

Мобільний додаток отримує данні з серверу в яких міститься інформація про кількість приміщень підключених до вентиляційної системи, показники складу повітря в кожному з приміщень, показники складу повітря на вулиці, та стан вентиляційної установки. Користувач може обрати конкретну кімнату для примусового увімкнення вентиляції.

Отримавши інформацію з серверу додаток аналізує показники повітря, присвоює рівень небезпеки для кожного, та присвоює рівень небезпеки для кожного приміщення підключеного до системи. Індикатори показників для кожного приміщення демонструють рівень небезпеки.

Опираючись на показники, додаток розподіляє потужність вентиляційної системи по приміщенням, та демонструє це користувачу. Інформація стосовно необхідності роботи системи та розподілення потужності посилається на сервер для виконання.

# 2 ВИКОРИСТАНІ ТЕХНІЧНІ ЗАСОБИ

Програмний модуль було протестовано на мобільному телефоні Apple iPhone 6, який працює на базі процесору A8 та має 2 Гб оперативної пам’яті. Розроблене програмне забезпечення є адоптованим до усіх пристроїв з операційною системою iOS 11 або новіше, що дозволяє запускати його на любому телефоні компанії Apple з операційною системою iOS 11 або новіше.